

DSD HW1 Report

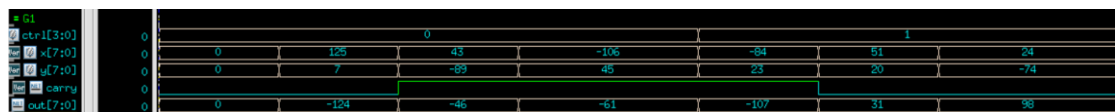
林軒霆 B11901132

1. ALU

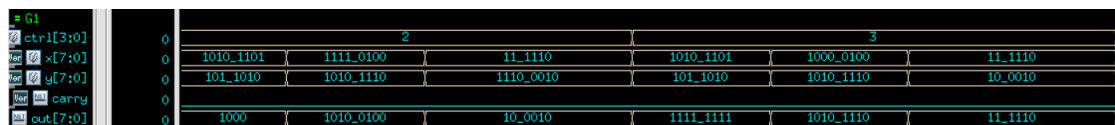
For the test bench in both ALU designs, I hand-crafted two to three test cases with correct answers for each function with manual calculation.

(a.) Assign

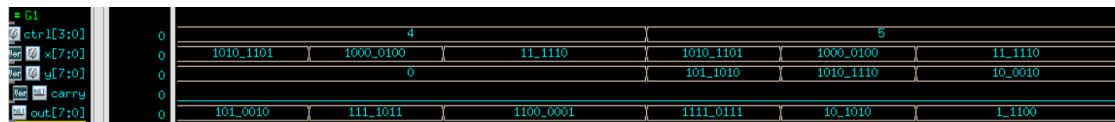
Signed addition & signed subtraction in decimal (note that the first case of addition produces overflow)



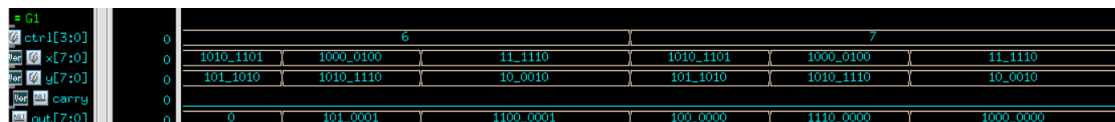
Bitwise and & bitwise or



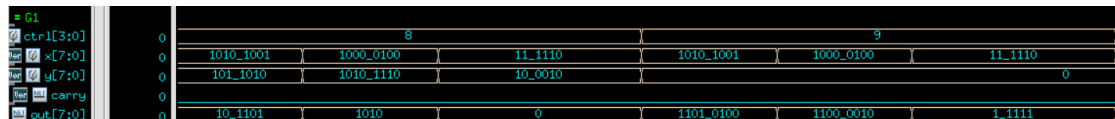
Bitwise not (y set to 8'b0) & bitwise xor



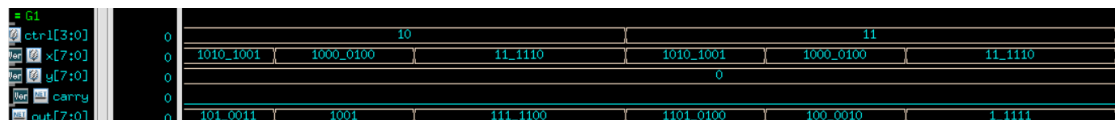
Bitwise nor & shift left logical variable



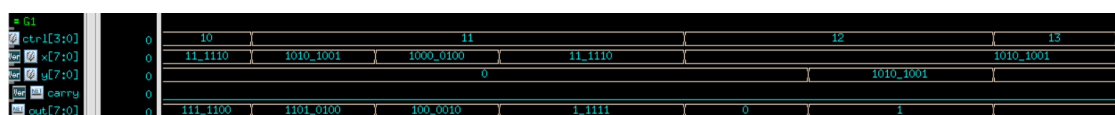
Shift right logical variable & shift right arithmetic (y set to 8'b0)



Rotate left (y set to 8'b0) & rotate right (y set to 8'b0)



Equal & NOP



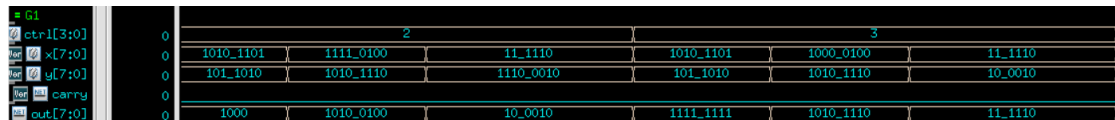
(b.) Always

Signed addition & signed subtraction in decimal (note that the first case of addition produces overflow)



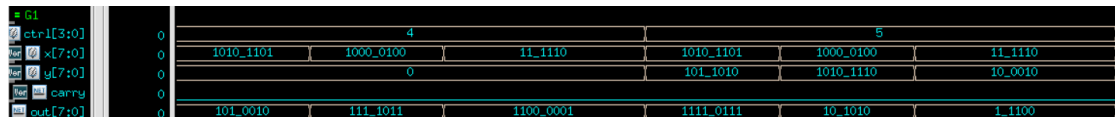
ctrl[3:0]	0	1	2	3	4	5	6	7
x[7:0]	0	125	43	-106	-84	51	24	
y[7:0]	0	7	-69	45	23	20	-74	
carry	0							
out[7:0]	0	-124	-46	-61	-107	31	98	

Bitwise and & bitwise or



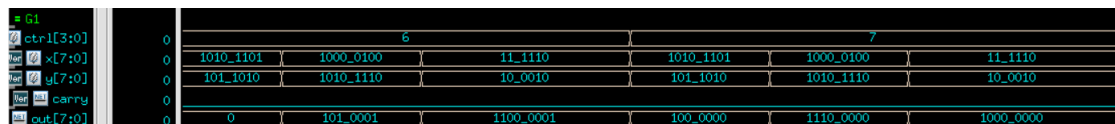
ctrl[3:0]	0	2	3	4	5	6	7
x[7:0]	1010_1101	1111_0100	11_1110	1010_1101	1000_0100	11_1110	
y[7:0]	101_1010	1010_1110	1110_0010	101_1010	1010_1110	10_0010	
carry	0						
out[7:0]	1000	1010_0100	10_0010	1111_1111	1010_1110	11_1110	

Bitwise not (y set to 8'b0) & bitwise xor



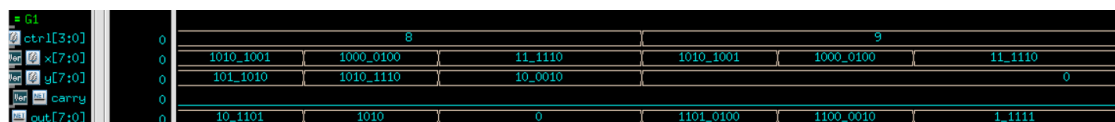
ctrl[3:0]	0	4	5	6	7	8	9
x[7:0]	1010_1101	1000_0100	11_1110	1010_1101	1000_0100	11_1110	
y[7:0]	0	0	0	101_1010	1010_1110	10_0010	
carry	0						
out[7:0]	101_0010	111_1011	1100_0001	1111_0111	10_1010	1_1100	

Bitwise nor & shift left logical variable



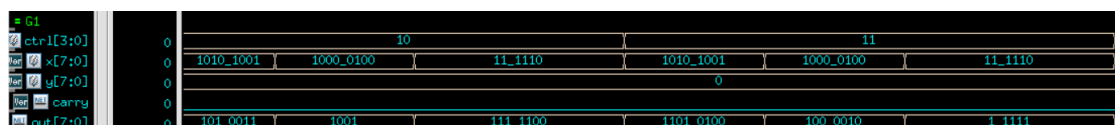
ctrl[3:0]	0	6	7	8	9	10	11
x[7:0]	1010_1101	1000_0100	11_1110	1010_1101	1000_0100	11_1110	
y[7:0]	101_1010	1010_1110	10_0010	101_1010	1010_1110	10_0010	
carry	0						
out[7:0]	0	101_0001	1100_0001	100_0000	1110_0000	1000_0000	

Shift right logical variable & shift right arithmetic (y set to 8'b0)



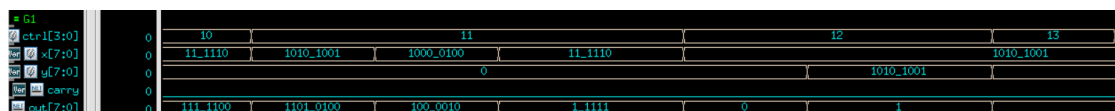
ctrl[3:0]	0	8	9	10	11	12	13
x[7:0]	1010_1001	1000_0100	11_1110	1010_1001	1000_0100	11_1110	
y[7:0]	101_1010	1010_1110	10_0010			0	
carry	0						
out[7:0]	10_1101	1010	0	1101_0100	1100_0010	1_1111	

Rotate left (y set to 8'b0) & rotate right (y set to 8'b0)



ctrl[3:0]	0	10	11	12	13	14	15
x[7:0]	1010_1001	1000_0100	11_1110	1010_1001	1000_0100	11_1110	
y[7:0]				0		1010_1001	
carry	0						
out[7:0]	101_0011	1001	111_1100	1101_0100	100_0010	1_1111	

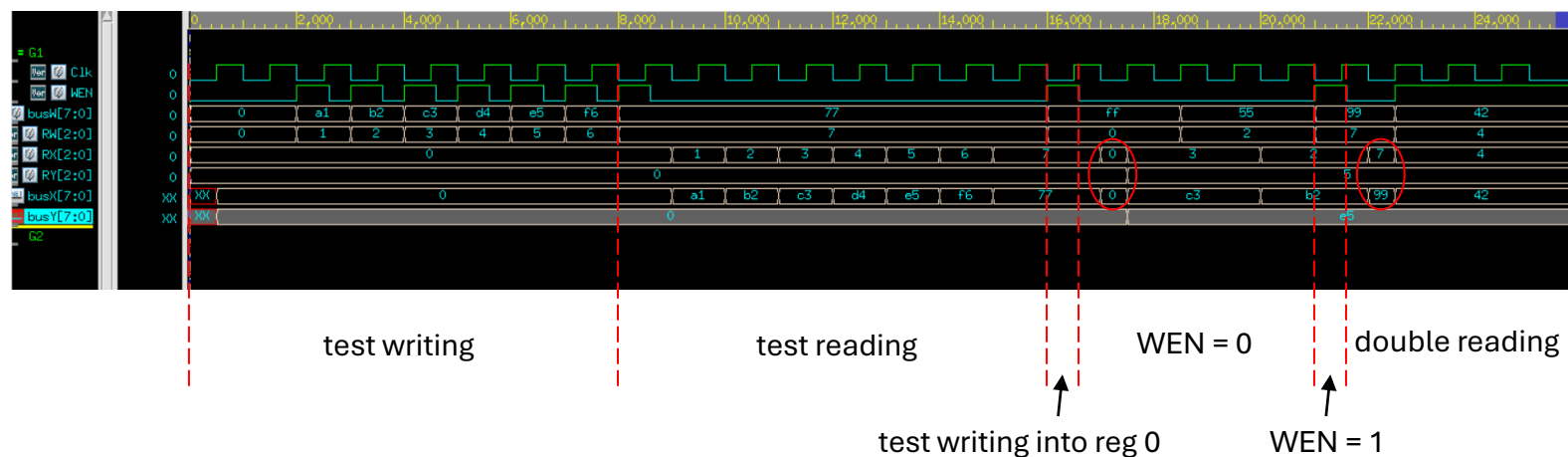
Equal & NOP



ctrl[3:0]	0	10	11	12	13	14	15
x[7:0]	11_1110	1010_1001	1000_0100	11_1110		1010_1001	
y[7:0]			0			1010_1001	
carry	0						
out[7:0]	111_1100	1101_0100	100_0010	1_1111	0	1	

2. Register File

The testbench contains several operations the register file may need to handle (such as writing, reading, overwriting, writing with WEN = 0 etc.), and I manually calculate the correct output to check correctness of implementation.



3. What I Found

(a.) Naming

While checking the simple calculator's functionality, there was a bug that made the outputs erroneous. After checking the individual modules again and a decent amount of time debugging, it turns out there was one letter in a wire of the calculator that was wrongly capitalized. This shows that proper naming and consistent naming style is important for efficiency and also code readability.

(b.) Latches

The first version of my ALU always block code contained latches. After Week 4 of the DSD class, I reviewed my code again and added default values in the combinational circuit to remove the latches. The original version worked and passed the testbench I wrote, but neither the testbench nor the compiler could detect the presence of latches. We need to develop good coding habits and pay special attention to the occurrence of latches.

(c.) Testbench

Manually calculating cases for the testbench is time-consuming while also prone to human errors, there should be better ways to create a testbench that can test all combination of inputs while keeping the code clean and correct. Using code like Python to create testbenches could be a promising method, all we have to do is to make sure the code that produces test cases are correct.

However, I have already finished writing the testbench after painstaking hours before coming up with this idea, I would try using other ways to produce testbenches in the future.